Crowdsource Image Classification Application for Machine Learning Systems

Alexander Cromer
Dept. of Computer Science, University of North Georgia, Dahlonega, Georgia, U.S.A.
*Corresponding author
Email address: cromeralec@gmail.com

***Abstract*— Machine Learning can require large datasets for training accurate and predictable systems. One of the issues for image-based classification systems is the time-consuming task of finding corresponding and specific datasets. This application is proposed as an easier method to not only store these datasets for use in machine learning, but to also give its users the ability to download specific searches using inclusion and exclusion searches, which can be absent in many of the image training sites. In order to minimize the strain of the system, I also proposed a method for image duplication detection and rejection to reduce the possibility of duplicates in a system allowing for less re-entries on a large scale.**

Keywords — Image storage; Data storage; Search Queries
Languages/API — Java, MySQL, JDBC, Javaxt

I. Introduction

There are many different types of machine learning systems. In the case of my application, I had a focus on data-driven systems where these applications rely on training data from positive datasets. I also wanted to allow the possibility for use of negative image datasets. With both of these specific sets of images, a machine learning algorithm can better understand how to classify or identify more accurately and precisely. When it comes to finding this data, the process can be lengthy. Many websites such as Open Images Dataset feature search methods for finding data but can be skewed to feature aspects that an operator might find undesirable. Images of birds will result in bird images, but may feature aspects such as trees, grass, waterfalls, leaves…etc. This is where crowdsourced collaboration can be extremely beneficial to generate large datasets [1]. Crowdsourcing can be very time consuming, especially with unintuitive or confusing systems. This application is designed to be simple and user friendly for both individuals who are generating the data as well as individuals who are sorting for specific data.

The application is also built to return very specific images, and if needed a collection of 'negative' images, which are images that the machine learning system should ignore either for training purposes or for testing to detect false positives.

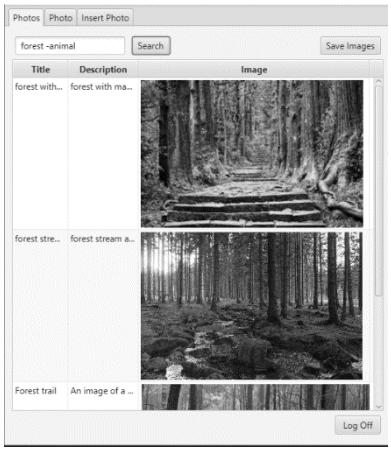
In the case of an applicable machine learning system such as the weapons detection project by David Anderson, this application would be beneficial to help reduce the number of false positives by allowing the training data to be varied between generic searches such as 'weapons', or for more specific searches such as a specific weapon model or to have exclusions of weapons such as 'swords', or 'shields' which may not be a priority.

I started off the project using Java and JavaFX for the GUI. I used the model-view-controller architectural pattern. The idea was to create a framework now, and if implementing a servlet in the future was needed, I could then modify the controller classes accordingly. The decision for using JavaFX came about due to its relatively straight forward documentation and simplicity, but also for its ease of accessibility implementation. The view builders that are standard in IntelliJ and Scene Builder for JavaFX feature button mapping for mouse-free use as well as section listing for help in organizing 'tab' movement. With over 600 million people being disabled in the population, making sure the application was accessible in at least a small area was considered a relatively high priority non-functional requirement. I started by creating some design documents to help layout my system such as the class diagram and an ER diagram which ended up needing to get changed as the development of the application progressed. The design documents helped to better understand my layout as I worked [6].

II. Data Storage

Adding data to a database was a necessary requirement. Using the Java Database Connectivity Driver I was able to use mySQL for persistent storage where I separated the database into three major parts: Image storage, color data, and image tagging. The image storage section held data such as the image metadata, blobs for the images themselves and descriptors such as title and description. The color data table held the RGB value of each of the image's sections. I decided to keep the color table denormalized, as this increases readability and performance especially since each image produces sixteen rows of data [3]. The tagging tables were a simple many-to-many relationship with a relationship "bridge" table to allow tags to be referenced from multiple images, rather than storing tags multiple times themselves. The application also gives the individual flexibility in tagging very specific information regarding an image such as details defined by the uploader as well as latitude, longitude, and camera model.

III. Searching

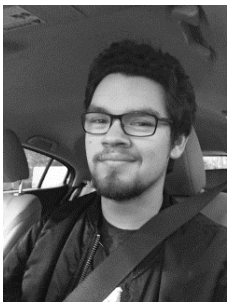


Search queries for the system were broken up into multiple parts and parsing the user query was a requirement to determine which of my designed search queries the system would perform. This was also added into the fact that the system had to pull data for opposite queries if the user requested it. Queries can be broken up into inclusion searches (forest), exclusion searches where the user can add a hyphen (-animal) to indicate that they do not want any results featuring that tag, or both. The system then combines every term, separates these tags into their respective types and then converts them to be submitted as a SQL query. The system allows an unlimited number of inclusion and exclusion tags to be included so the results can be very personalized.

IV. Duplication Detection

I knew starting this project that checking for image duplicates would be relatively valuable. Images can differ in bit rates, resolutions, and shades in color due to compression, resizing, or re-exportation. In order to combat these possibilities, I wanted to include an algorithm that could be used to detect duplicates. The detector first takes the file name for the image and determines if the image is an even number. If the vertical or horizontal row/column of the image is not an even number, then it removes the outer most row of pixels. The application then uses a quadruple for-loop to first divide the image into sixteen sections. Within each section the application takes each pixel's red, green, and blue value and stores it to calculate the average. These averages get stored into an array and then sent to the image checker to make sure it is not a duplicate.

The issue at this point was due to the lack of pixels in smaller resolutions as there were minute variations in average color which could be exacerbated from lost highlights or shadows. To remedy this, I implemented an option of having a tolerance value. This tolerance value is used in conjunction with a given section's average to then create a range that can be searched for in the database. A value of 120 with a tolerance value of '5', would search the database for an image through a range between 115 and 125. The higher this tolerance value, the easier it is for the system to find false positives, and a lower tolerance value increases the likely-hood of a duplicate image being missed when an image may verge on having more extreme shifts in color or resolution, albeit very unlikely. If resolution is the main combating factor, a tolerance of 1-2 is recommended to combat changes in color shift, while extremely compressed images may require a tolerance between 3-4.



72	66	121	61
83	73	126	66
88	7**1**	116	66
60	129	89	67
67	102	81	7**0**
72	93	74	70
49	188	151	59
54	126	110	65
61	10**1**	80	68
60	7**1**	87	39
67	67	83	47
77	67	7**3**	55

(a)

72	66	121	61
83	73	126	66
88	7**2**	116	66
60	129	89	68
67	102	81	7**1**
72	93	74	70
49	188	151	59
54	126	110	65
61	10**2**	80	68
60	7**2**	87	39
67	67	83	47
77	67	7**4**	55

(b)

Average RGB values for the sixteen different sections with bolded numbers representing discrepancies from the change in resolution. (a) A high-resolution image (b) A low-resolution image

V. Changes for the Future

Going forward, I would implement a few additional features. One would be a client-server-based approach to not only reduce strain on a client computer, but to allow image duplication detection to occur in the background while a user adds photos. The next would be to implement a folder explorer to allow batch file inputs; This would cut down on some of the overhead of having to re-open the file chooser. The next feature to add would be for the system to operate with the use of 'alias' tags. This would include generalizations of more specifically defined tags but also to help with images that are undefined, as well as predictions for any additional tags. EG: A image tagged with 'Ford' would be returned under a query for 'Vehicle', regardless if the image was tagged as such. This would significantly reduce the effort required to manually tag data that other systems suffer from [4]. Finally, the last additional change I would make would be to implement a drawing canvas for outlining sections of images with more significant items through either a square or more complex polygons. With the canvas addition, the locations for the pixels that are important would be stored and that queried image would be cropped from the original.

VI. Conclusion

Image classification can be an expensive and lengthy process, not just for smaller companies, but as well as larger companies where small inefficiencies in the process can slow the task of image sorting over time. In my application, I have attempted to relieve issues such as possible redundancies in image classification due to the possibility of a duplicate, accessibility concerns, and results from very specific searches. In databases featuring over one-hundred thousand images this detection can be alleviating to both reducing the amount of storage needed, while also allowing individuals to process other significant images instead. Not only do the accessibility features provide benefits for individuals with disabilities, but the ease of customization for button mapping and mouse-less navigation may make image classifying a faster process to those who are not. Overall, I feel that I have accomplished what I set out to do this semester.

References

[1] Y. Roh, G. Heo, S. Whang, "A Survey on Data Collection for Machine Learning: a Big Data - AI Integration Perspective,", pp. 1-8, 2018.

[2] A. Marcus, A. Parameswaran, "Crowdsourced Data Management: Industry and Academic Perspectives," in foundations and Trends in Databases, pp. 1-161, 2015.

[3] G. Sanders, S. shin, "Denormalization Effects on Performance of RDBMS," for Hawaii International Conference on System Sciences, pp. 1-3, 2001.

[4] O. Russakovsky, J. Deng, H. Su, et al., "Imagenet large scale visual recognition challenge," International Journal of Computer Vision, vol. 115, no. 3, pp. 1-5, 2015.

[5] P. Kotzé, M. Eloff, A. Adesina-Ojo, "Accessible Computer Interaction for People with Disabilities" for the International Conference on Enterprise Information Systems, pp. 98-105, 2004.

[6] B. Bruegge, A. Dutoit, "Object-Oriented Software Engineering using UML, Patterns, and Java Third Edition", pp. 30-76, 2010.